



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

**(Laboratorio di)
Amministrazione di sistemi**

Esecuzione e monitoraggio dei servizi

Marco Prandini

Dipartimento di Informatica – Scienza e Ingegneria

Demoni

- **Nel gergo Unix, i servizi di sistema**
 - avviati e arrestati automaticamente
 - disconnessi da qualsiasi terminale
 - eseguiti a nome di utenti specifici senza accesso a shell interattiva
- **Operazioni comuni**
 - configurare le condizioni per il loro avvio e arresto automatico
 - esaminarne lo stato complessivo
 - tracciarne l'esecuzione
 - avviarli, arrestarli o inviare loro segnali manualmente



Gestione dei processi

■ Dopo un'installazione "minimale"...

milk:~# ps aux

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	1948	468	?	Ss	May15	0:02	init [2]
[... kernel processes ...]										
root	1753	0.0	0.0	2704	392	?	S<s	May15	0:00	udevd --daemon
daemon	2953	0.0	0.0	1688	408	?	Ss	May15	0:00	/sbin/portmap
root	3231	0.0	0.0	1624	568	?	Ss	May15	0:26	/sbin/syslogd
root	3237	0.0	0.0	1576	340	?	Ss	May15	0:00	/sbin/klogd -x
bind	3251	0.0	0.1	39732	1964	?	Ssl	May15	0:00	/usr/sbin/named
root	3266	0.0	0.0	39500	944	?	Ssl	May15	0:00	/usr/sbin/lwres
root	3339	0.0	0.0	1572	444	?	Ss	May15	0:00	/usr/sbin/acpid
103	3344	0.0	0.0	2376	760	?	Ss	May15	0:00	/usr/bin/dbus-d
106	3352	0.0	0.1	6116	1972	?	Ss	May15	0:03	/usr/sbin/hald
root	3353	0.0	0.0	2896	716	?	S	May15	0:00	hald-runner
106	3359	0.0	0.0	2016	472	?	S	May15	0:00	hald-addon-acpi
106	3367	0.0	0.0	2020	480	?	S	May15	0:00	hald-addon-keyb
root	3387	0.0	0.0	1808	360	?	S	May15	14:15	hald-addon-stor
root	3414	0.0	0.0	1864	396	?	Ss	May15	0:00	/usr/sbin/dhcdb
root	3421	0.0	0.1	3984	1164	?	Ss	May15	0:00	/usr/sbin/Netwo
avahi	3433	0.0	0.1	2936	1424	?	Ss	May15	4:14	avahi-daemon: r
avahi	3434	0.0	0.0	2552	180	?	Ss	May15	0:00	avahi-daemon: c
root	3441	0.0	0.0	2908	536	?	Ss	May15	0:00	/usr/sbin/Netwo
root	3457	0.0	0.0	1752	452	?	Ss	May15	0:02	/usr/sbin/inetd
root	3477	0.0	0.0	4924	512	?	Ss	May15	0:02	/usr/sbin/sshd
ntp	3507	0.0	0.0	4144	764	?	Ss	May15	0:00	/usr/sbin/ntpd
root	3521	0.0	0.0	1976	724	?	Ss	May15	0:02	/sbin/mdadm --m
daemon	3540	0.0	0.0	1828	280	?	Ss	May15	0:00	/usr/sbin/atd
root	3547	0.0	0.0	2196	720	?	Ss	May15	0:00	/usr/sbin/cron
root	3590	0.0	0.0	1572	372	tty2	Ss+	May15	0:00	/sbin/getty 384
root	3591	0.0	0.0	1576	372	tty3	Ss+	May15	0:00	/sbin/getty 384
root	3592	0.0	0.0	1572	372	tty4	Ss+	May15	0:00	/sbin/getty 384
root	3593	0.0	0.0	1572	372	tty5	Ss+	May15	0:00	/sbin/getty 384
root	3595	0.0	0.0	1576	372	tty6	Ss+	May15	0:00	/sbin/getty 384

Gestione dei processi

- Anche se tutti questi processi fossero utili, sarebbe importante
 - sapere che origine hanno
 - sapere come terminarli, evitando che ricompaiano
 - **processi inutili consumano risorse e offrono opportunità di attacco**
- Banali e fondamentali:
 - *man* è il vostro migliore amico, seguito da Internet.
 - *ps, top, kill, ...* sono efficaci per individuare e risolvere problemi istantanei, ma non garantiscono che non si ripresenteranno
- Ci sono tre fonti primarie di processi (oltre agli utenti)
 - Pianificatori periodici e sporadici
 - Demoni di gestione degli eventi
 - **Procedure di avvio del sistema**

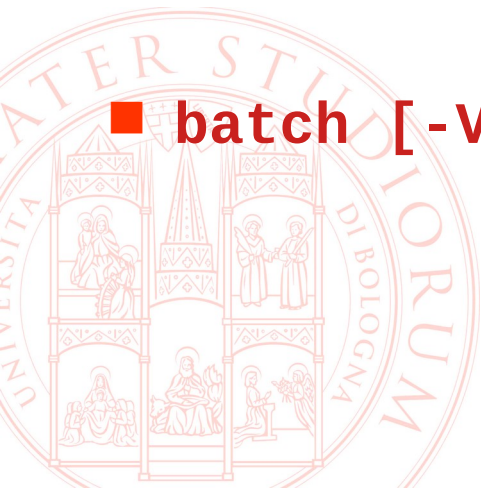


Esecuzioni pianificate

- L'esecuzione periodica di programmi è compito di **crond**
 - ogni utente ha la propria *cron table* (*crontab*),
 - guardate in **/var/spool/cron** per trovarle
 - i task di sistema sono spesso raccolti in **/etc/crontab**
 - tipicamente preconfigurato per l'esecuzione di script a periodicità di uso comune
 - **/etc/cron.hourly**, **/etc/cron.daily**, **/etc/cron.weekly**, **/etc/cron.monthly**
 - **/etc/crontab** si può editare direttamente, per le tabelle utente meglio usare **crontab -e [-u username]**
- L'esecuzione singola in un istante preciso è compito di **atd**
 - **atq** per elencare i job in attesa
 - **atrm** per rimuoverli

Esecuzione posticipata - at

- **atd** è un demone che gestisce code di compiti da svolgere in momenti prefissati. L'interfaccia ad **atd** consiste di 4 comandi:
- **at [-V] [-q queue] [-f file] [-mldbv] TIME**
pianifica un comando al tempo TIME
- **atq [-V] [-q queue] [-v]**
elenca i comandi in coda
- **atrm [-V] job [job...]**
rimuove comandi dalla coda
- **batch [-V] [-q queue] [-f file] [-mv] [TIME]**
esecuzione condizionata al carico



Esecuzione posticipata - at

- Se non viene specificato un file comandi per at o batch, verrà usato lo standard input.
- La specifica dell'ora è flessibile e complessa. Per una definizione completa si veda la documentazione in **/usr/share/doc/at/timespec**
- Alcuni esempi:

```
echo 'wall "sveglia"' | at 08:00
```

```
echo "$HOME/bin/pulisci" | at now + 2 weeks
```

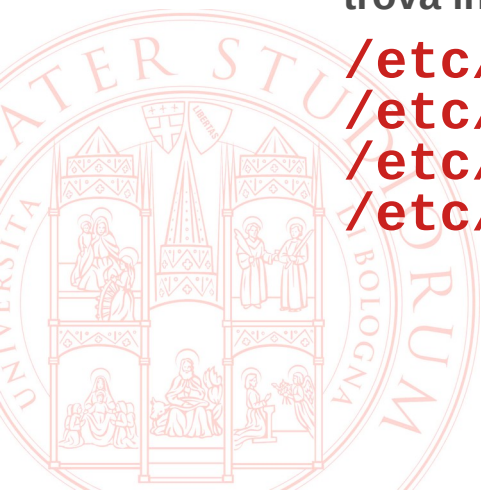
```
echo "$HOME/bin/auguri" | at midnight 31.12.2021
```



Esecuzione periodica - cron

- **crond** è un demone che esamina una serie di file di configurazione ogni minuto, e determina quali compiti specificati nei file debbano essere eseguiti.
- I file di configurazione (**crontab**) sono distinti in due insiemi:
 - Uno per utente (**/var/spool/cron/crontabs/<utente>**)
 - si visualizza / edita / sostituisce con
crontab -l / crontab -e / crontab <nuova_tab>
 - System-wide (**/etc/crontab**)
 - Solitamente quest'ultimo non fa altro che richiamare l'esecuzione di tutto ciò che trova in alcune directory:
**/etc/cron.hourly/
/etc/cron.daily/
/etc/cron.weekly/
/etc/cron.monthly/**

ha un campo in più rispetto ai file personali per indicare a nome di che utente eseguire ogni task configurato



Esecuzione periodica - cron

- Ogni crontab contiene un elenco di direttive nella forma
MINUTO ORA G.MESE MESE G.SETTIMANA **<comando>**

Es.

*	*	27	*	*	\$HOME/bin/paga
30	8-18/2	*	*	1-5	\$HOME/bin/lavora
00	00	1	1	*	/usr/sbin/auguri
30	4	1, 15	*	6	/bin/backup

- L'azione è eseguita quando l'ora corrente corrisponde a tutti i selettori di una riga (campi in AND logico)
- **ECCEZIONE**: se sono specificati (diversi da *) entrambi i giorni (settimana e mese), i due campi sono considerati in OR logico
 - nell'esempio, il backup viene eseguito ogni mese il giorno **1** + il giorno **15** + ogni **sabato**, alle 4:30

Event managers / IPC systems

- Dbus è un'architettura di Inter-Process Communication
 - Nata per uniformare la comunicazione tra elementi delle interfacce desktop
 - Curiosate in `/etc/dbus-1/` per vedere i file di configurazione
 - In `/etc/dbus-1/event.d` sono collocati gli script di avvio dei sottosistemi gestiti
- Udev ha rimpiazzato devfs come **event manager** per la creazione istantanea dei device special file quando un nuovo dispositivo viene connesso; ora è parte di systemd
 - In `/etc/udev/rules.d` sono configurate le regole evento → azione
 - Es. **70-persistent-net.rules**

```
# PCI device 0x10ec:0x8168 (r8169)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="d0:67:e5:18:d9:e4", ATTR{dev_id}=="0x0",
ATTR{type}=="1", KERNEL=="eth*", NAME="eth0"
```

alla comparsa nel subsystem `net` di una scheda con `MAC=d0:67:e5:18:d9:e4` le assegna nome `eth0`

Inizializzazione e attività in background (demoni)

■ *init* è il primo processo avviato dal kernel

- Gestisce i *runlevel*
 - stati di funzionamento del sistema definiti dal sottoinsieme di servizi attivi
- Orchestra la sequenza corretta di eventi per raggiungere un runlevel
- Intercetta e gestisce alcuni eventi
 - es. ctrl-alt-canc, terminazione anomala di processi,
- Spegne il sistema in modo ordinato

■ Tre varianti principali

- (storico) SystemV-style initialization
- Upstart (Canonical, 2006-2014)
- Systemd (ispirazione RedHat, 2010-oggi)

utile da conoscere
per l'attuale mix
imprevedibile di
distribuzioni moderne
e tradizionali



sysvinit

■ `/sbin/init` dell'originale SystemV Unix

- configurato dal file `/etc/inittab`
- `inittab` specifica il default runlevel
 - `id:2:initdefault:`
- ma se la keyword `single` viene passata come parametro al kernel dal boot loader, questo settaggio è scavalcato e il sistema parte in *single user mode* (runlevel 1)
 - `~~:S:wait:/sbin/sulogin`
- `init` avvia i virtual terminal e i gestori delle console su linea seriale (può sembrare un arcaismo, ma nel mondo IoT è tornato alla ribalta)
 - `1:2345:respawn:/sbin/getty 38400 tty1`
 - `2:23:respawn:/sbin/getty 38400 tty2`
 - ...
 - `T0:23:respawn:/sbin/getty -L ttyS0 9600 vt100`
 - `T3:23:respawn:/sbin/mgetty -x0 -s 57600 ttyS3`

processi avviati da sysvinit

- *init* è in senso astratto responsabile per tutti i processi che girano sul sistema, ma in particolare due attività possono essere direttamente ricondotte ad esso
 - linee del tipo

```
l0:0:wait:/etc/init.d/rc 0
```

pilotano il processo di avvio *System-V-style*

- wait = esecuzione sequenziale
- quando il runlevel obiettivo è 'N', **rc** esegue
 - ogni programma con nome che inizia per 'S' in **/etc/rcN.d/** col parametro **start**
 - ogni programma con nome che inizia per 'K' in **/etc/rcN.d/** col parametro **stop**
- per evitare l'inutile duplicazione degli script di avvio e arresto dei demoni, questi sono tutti raccolti in **/etc/init.d/**, e symlinked dalle 7 directory **/etc/rcN.d/**

- linee del tipo

```
x:5:respawn:/usr/X11/bin/gdm
```

avviano il programma specificato come 4° campo, e *init* monitora il processo per riavviarlo (respawn) se termina

Controllo del sistema con sysvinit

- Configurazione persistente (applicata automaticamente all'avvio)
 - **chkconfig** (RedHat) o **update-rc.d** (Debian) configurano i runlevel gestendo i symlink nelle 7 directory
- Verifica del runlevel attivo
 - **runlevel**
 - restituisce il precedentemente attivo e l'attuale
- Cambio di runlevel
 - **telinit N**
- Avvio, arresto e verifica dello stato dei singoli servizi
 - **/etc/init.d/scriptname {start|stop|status}**
 - supportati da alcuni script **reload, restart, condrestart, ...**



Upstart (principalmente Ubuntu)

- Un rimpiazzo per *init* basato sulla logica a eventi
 - Inizializzazione dei sottosistemi parallela e non bloccante
 - Gestione omogenea di tutti gli eventi asincroni
 - Aggiunta e rimozione di hardware
 - Avvio e arresto di processi
 - Inizializzazione multi-stadio (es. rilevazione hardware → caricamento firmware → attivazione device → rilevazione delle caratteristiche del device)
 - In prospettiva, integrazione dei pianificatori (cron, at)
- Distribuzioni principali che lo adotta(va)no
 - Ubuntu 6.10 – 14.10
 - Fedora 9 – 14
 - Debian (opzione)
 - Nokia's Maemo platform
 - Palm's WebOS
 - Google's Chromium OS
 - Google's Chrome OS

Qualche concetto di base su upstart

■ Filosofia (dal sito):

- Task e Servizi sono avviati e arrestati in seguito a eventi
- Il completamento di un avvio/arresto genera a sua volta un evento
- Gli eventi possono essere ricevuti da qualsiasi processo sul sistema
- I Servizi possono essere riavviati se terminano inaspettatamente
- La supervisione e il riavvio di un demone è gestita anche nel caso sia un processo figlio separato dal progenitore
- La comunicazione avviene via D-Bus

■ Operativamente

- La directory **/etc/init** contiene un file per definire ogni attività
- Il demone **init** continua ad essere l'orchestratore del sistema
 - ogni modifica ai file di configurazione è rilevata via inotify e applicata in tempo reale
- Il comando **initctl** interagisce con le attività mandando segnali appropriati (documentati nei sorgenti in **event.h**) a **init** (via sotto-comandi):
 - **start / stop / status**
 - **list / emit / reload-configuration**

Systemd (ispirato da RedHat – ora molto diffuso)

■ Che aspetti affronta systemd?

- Dipendenze tra servizi
- Avvio a richiesta di servizi
- Logging precoce
- Conservazione dell'output dei demoni
- Tracciamento dei cgroup (per controllo preciso risorse hardware)
- Tracciamento e gestione dei mount point
- Snapshots di sistema e loro ripristino
- Gestione delle impostazioni globali come hostname, locale, ecc.
- Ambiente deterministico di esecuzione dei servizi
- Aggiornamenti del sistema offline (al riavvio)
- Processo di boot più rapido e senza shell interattive



Systemd

■ Systemd si propone di sostituire

- init (etc.)
- udev
- pm-utils
- inetd
- acpid
- crond/atd
- ConsoleKit
- automount
- watchdog
- syslog



Systemd – termini

- Diversi tipi di **[control] unit** i cui nomi seguono la convenzione **name.type**
- **type** può essere:
 - **Service**: controllo e monitoraggio dei demoni
 - **Socket**: attivazione di canali IPC di ogni tipo (file, net socket, Unix socket)
 - **Target**: gruppo di unit che **rimpiazza il concetto di runlevel**
 - **Device**: punti di accesso ai dispositivi, creati dal kernel in seguito a interazioni con l'hardware
 - filesystem-related: **Mounts**, **Automounts**, **Swap**
 - **Snapshots**: stato salvato del sistema
 - **Timers**: attività legate al tempo (→ cron, at)
 - **Paths**: monitoraggio del contenuto di una directory via inotify
 - **Slices**: gestione delle risorse via cgroup
 - **Scopes**: raggruppamento di processi per miglior organizzazione

Systemd – dove trovare le definizioni delle unit

- “libreria” di definizioni di riferimento
 - `/lib/systemd/system/`
- File forniti dai mantainer dei diversi pacchetti software
 - Quasi sempre link alle definizioni di riferimento
 - `/usr/lib/systemd/system/`
- File con le personalizzazioni
 - prioritari rispetto alle definizioni di sistema sopra elencate
 - `/etc/systemd/system`



Systemd – avvio e arresto dei servizi

■ Controllo a run time dei servizi

- `systemctl {start|stop|status|restart|reload} servicename`
 - ...intuitivo
 - output molto descrittivo dello stato
 - stato corrente ed elenco dei passi fatti per raggiungerlo
 - process tree
 - righe di log rilevanti
 - con `-H [hostname]` si connette a un host remoto via ssh

■ Cosa fanno in realtà? Vedremo in pratica i dettagli, ma in sintesi

- Nelle unit sono definiti i comandi da eseguire attraverso parametri di configurazione (`ExecStart`, `ExecReload`, `ExecStop`)
 - `restart` è semplicemente `stop` seguito da `start`
- systemd tiene traccia dei processi avviati con `start`, in modo che i loro PID possano essere usati come parametri nei comandi `reload/stop`
- `stop`, oltre a eseguire (l'eventuale) comando specificato, di default manda `SIGTERM` al processo, seguito da `SIGKILL` dopo un timeout. Moltissime varianti configurabili: `man 5 systemd.kill`

Systemd – boot e shutdown

- Le operazioni descritte alla slide precedente sono **volatili**
 - si impartisce il comando
 - si ottiene l'effetto
 - nulla cambia nella **configurazione** del sistema
- Per automatizzare avvio al boot e arresto allo shutdown si utilizzano invece
 - **systemctl {enable|disable|mask|unmask} servicename**
 - **disable** lascia disponibile la possibilità di usare manualmente **start**
 - **mask** "neutralizza" l'intera definizione della unit, impedendo anche il controllo manuale
 - questi comandi non hanno alcun effetto immediato
 - **l'effetto sulla configurazione del sistema è persistente**



Systemd – verifica della configurazione

■ Solo qualche esempio

- **systemctl list-units**
 - mostra tutte le *unit* gestite (di tutti i tipi elencati!)
- **systemctl -t type**
 - es.: **systemctl -t timers**
 - mostra tutte le unit attive del tipo specificato
- **systemctl list-unit-files [-t type]**
 - es.: **systemctl list-unit-files -t services**
 - mostra tutte le unit installate del tipo specificato
- **systemctl --state state**
 - es.: **systemctl --state failed**
 - mostra tutte le unit che si trovano nello stato specificato



Cheat sheet

service e *systemctl* sono stati introdotti dai rispettivi sistemi ma sono wrapper retrocompatibili (in alcuni sistemi c'è un misto di demoni gestiti in 2 o tutti e 3 i modi!) es. se *systemctl start name* non trova la unit name, prova *service name start*, così come questo proverebbe */etc/init.d/name start* in caso di assenza di configurazione upstart

	SysVinit (Debian) (RedHat)	Upstart	Systemd
Start service	<code>/etc/init.d/name start</code>	<code>service name start</code>	<code>systemctl start name</code>
Stop service	<code>/etc/init.d/name stop</code>	<code>service name stop</code>	<code>systemctl stop name</code>
Status check	<code>/etc/init.d/name status</code>	<code>service name status</code>	<code>systemctl status name</code>
Enable service start at boot	<code>update-rc.d name enable</code> <code>chkconfig name on</code>	<code>rm /etc/init/name.override</code>	<code>systemctl enable name</code>
Inhibit service start at boot	<code>update-rc.d name disable</code> <code>chkconfig name off</code>	<code>echo manual ></code> <code>/etc/init/name.override</code>	<code>systemctl disable name</code>
List installed services	<code>ls /etc/init.d</code> <code>chkconfig --list</code>	<code>service --status-all &&</code> <code>initctl list</code>	<code>systemctl list-unit-files</code> <code>-t services</code>
List services starting at boot	<code>ls /etc/rcX.d/S*</code> <code>chkconfig --list </code> <code>grep X:on</code>	<i>Give up and upgrade to systemd.</i>	<code>systemctl list-unit-files</code> <code>-t services</code> <code>--state=enabled</code>

X = runlevel di default

Assunzione standard:
i servizi installati sono
configurati per partire al boot

Systemd – avvio

■ I runlevel sono rimpiazzati dai target

/etc/inittab non è più utilizzato

- il target di default è visualizzabile/impostabile con

systemctl get-default

systemctl set-default [target]

- es.: **systemctl set-default graphical.target**

■ Equivalenze

- Esplorate **/lib/systemd/system**

SysV Runlevel	systemd Target	Notes
0	runlevel0.target, poweroff.target	Halt the system.
1, s, single	runlevel1.target, rescue.target	Single user mode.
2, 4	runlevel2.target, runlevel4.target, multi-user.target	User-defined/Site-specific runlevels. By default, identical to 3.
3	runlevel3.target, multi-user.target	Multi-user, non-graphical. Users can usually login via multiple consoles or via the network.
5	runlevel5.target, graphical.target	Multi-user, graphical. Usually has all the services of runlevel 3 plus a graphical login.
6	runlevel6.target, reboot.target	Reboot
emergency	emergency.target	Emergency shell

Systemd – avvio

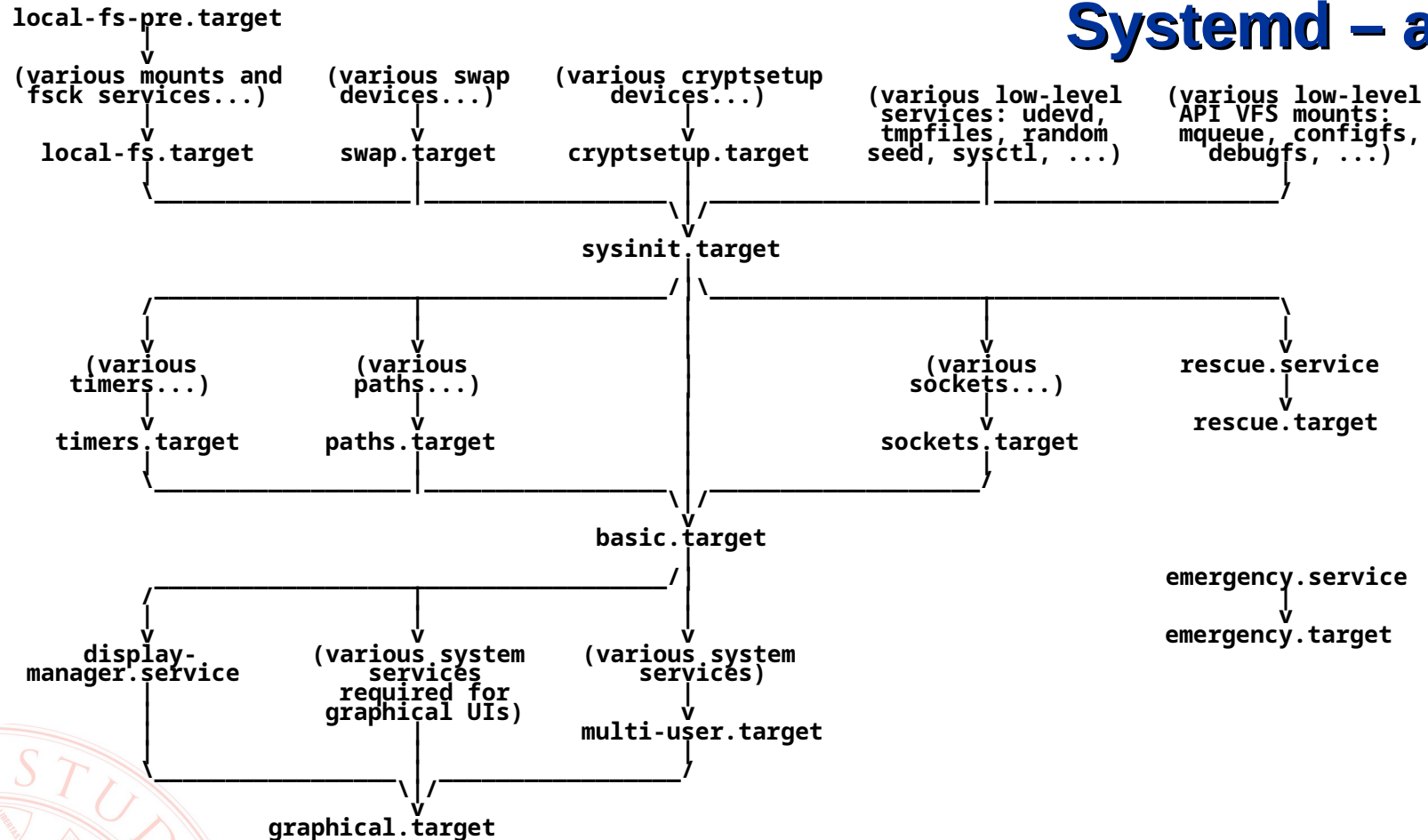
■ Cosa fa un target? Dalla man page **systemd.target (5)**:

“Target units [...] exist merely to **group units via dependencies** (useful as boot targets), and to **establish standardized names** for synchronization points used in dependencies between units.”

■ Dipendenze = automazione robusta

- Sysvinit = sequenziale → lento, nessuna gestione errori
- Systemd = parallelo condizionato → ogni unit parte non appena sono rispettati i vincoli espressi dalle direttive:
 - **Requires** – elenco di altre unit da avviare quando questa è avviata/fermata: se l'avvio di tali unit fallisce, questa viene arrestata; si può configurare la relazione temporale (dopo, prima, simultaneamente)
 - **Wants** – versione più soft di Requires (il fallimento delle dipendenze non blocca l'avvio di questa unit)
 - **Conflicts** – vincolo negativo per rendere unit mutuamente esclusive
 - **OnFailure** – unit da avviare quando questa fallisce
 - **RequiredBy** / **WantedBy** – **crea automaticamente entry Requires/Wants nelle unit elencate quando questa viene installata**
 - **Restart** – riavvia il servizio in caso di terminazione; è l'equivalente del respawn di inittab, ma con una varietà ricca di ulteriori sotto-parametri per controllare sotto quali condizioni effettivamente eseguire il riavvio

Systemd – avvio



■ Unit speciali

– Alcune unit sono predefinite con nomi fissi e funzioni fondamentali

- Principalmente target, e alcune slice (vedi `systemd.special(7)` e `bootup(7)`)
- Es. punti di controllo della sequenza di boot, che punta a `default.target`
- `default.target` sarà un link a uno dei "veri" target disponibili

Systemd - scrivere uno unit file per un servizio

- La quantità di opzioni di configurazione è enorme
 - `man 5 systemd.service`
 - <https://www.freedesktop.org/software/systemd/man/systemd.service.html>
- Gli elementi veramente indispensabili però sono pochi

[Unit]

<code>Description=</code>	<i>Descrizione</i>
<code>Requires/Wants=</code>	<i>da chi dipende questa unit</i>
<code>Documentation=</code>	<i>es. <code>man:rsyslogd(8)</code> o URL o altro</i>

[Service]

<code>Type=</code>	<i>tipo di avvio</i>
<code>ExecStart=</code>	<i>comando da lanciare all'avvio</i>
<code>ExecStop=</code>	<i>comandi (opzionali) per stop</i>
<code>ExecReload=</code>	<i>comandi (opzionali) per reload</i>
<code>Restart=</code>	<i>reazione (opzionale) alla terminazione</i>

[Install]

<code>WantedBy=</code>	<i>chi dipende da questa unit</i>
<code>Alias=</code>	<i>nome con cui è nota a systemd</i>

Systemd - scrivere uno unit file (dipendenze)

■ Handling dependencies

- Le dipendenze possono (devono) essere risolte direttamente in fase di design dei files di unit.

■ Esempio:

- prima di poter avviare la unit A, la unit B deve essere in stato di running
 - basta aggiungere **Requires=B** e **After=B** alla sezione **[Unit]** di A
 - se la dipendenza fosse opzionale allora si userebbe **Wants** al posto di **Requires** (tenta di avviare B ma in caso di fallimento non blocca la partenza di A)
- Notate che **Wants** e **Requires** non implicano **After**, questo vuol dire che senza **After** la semantica sarà la partenza in parallelo delle unit, e verifica a posteriori del vincolo

■ Le dipendenze sono tipicamente inserite nei servizi e non nei Target.



Systemd - scrivere uno unit file (tipo 1/2)

Ci sono alcuni tipi diversi di start-up da considerare quando si scrive un file unit personalizzato. Questo è specificato con il parametro **Type** nella sezione **[Service]**

■ **Type=simple** (default)

- systemd considera il servizio avviato con successo non appena ha forkato un processo figlio per eseguire il comando **ExecStart** (anche se poi tale comando dovesse fallire!). Il processo non deve forkare. Non usare questo tipo se altri servizi devono essere ordinati da questo servizio, a meno che siano attivati tramite socket.

■ **Type=forking**

- systemd considera il servizio partito una volta che il processo avviato con **ExecStart** esegue una propria fork e il genitore esce. È tipicamente usato per riutilizzare un “classico” demone Unix. È raccomandabile usare l’opzione **PIDFile=** affinché systemd possa tracciare il processo principale.

■ **Type=oneshot**

- utile quando abbiamo uno script/job da lanciare una volta sola e poi uscire. Si può settare **RemainAfterExit=yes** così che systemd possa considerare il servizio come attivo dopo la sua uscita.

Systemd - scrivere uno unit file (tipo 2/2)

■ Type=notify

- Identico a **Type=simple**, ma systemd si aspetta che quando il servizio è effettivamente pronto gli mandi un segnale via **sd_notify(3)** o simile

■ Type=dbus

- il servizio è considerato ready quando uno specifico **BusName** compare nel Dbus.

■ Type=idle

- systemd rimanderà l'avvio di servizi di questo tipo (per max. 5 secondi) fino a che tutti gli altri jobs saranno smistati. Oltre a questo non cambia molto con **Type=Simple** ed è unicamente utile per “tenere in ordine” i messaggi su console



Systemd - scrivere uno unit file (start, stop e reload)

- Tutti i servizi usano **ExecStart** per avviare un comando
 - se è una riga di shell, va eseguita con **sh -c 'comandi'**
 - il PID verrà memorizzato da systemd in **\$MAINPID**
- Se si vuole supportare il tipico comportamento di reload, il processo deve essere in grado di gestire un segnale lanciato da systemd quando si esegue **systemctl reload <servizio>**
 - nella configurazione avremo ad esempio **ExecReload=kill -HUP \$MAINPID**
- Per un servizio persistente (tipo **simple**) è molto comune non specificare **ExecStop**
- I servizi di tipo **oneshot** tipicamente configurano qualche aspetto del sistema quando avviati e ripristinano lo stato precedente quando arrestati
 - avremo quindi ad esempio
ExecStart=/usr/bin/mio-configuratore
ExecStop=/usr/bin/mio-de-configuratore

Systemd - scrivere uno unit file (monitoraggio)

- Con l'opzione **Restart** si può delegare systemd a monitorare il processo avviato
 - Se il processo termina per cause diverse da **systemctl stop** o non è reattivo
 - per questo tipo di controllo bisogna gestire i watchdog
- systemd lo riavvierà o meno, a seconda della combinazione tra il valore di **Restart** e la causa di malfunzionamento rilevata

Restart settings Exit causes	no	always	on- success	on- failure	on- abnormal	on- abort	on- watchdog
Clean exit code or signal		X	X				
Unclean exit code		X		X			
Unclean signal		X		X	X	X	
Timeout		X		X	X		
Watchdog		X		X	X		X

Systemd - scrivere uno unit file

- Una volta scritta la vostra unit Systemd bisogna effettuare un soft reload caricando dal filesystem le varie Unit e rigenerando l'albero delle dipendenze
 - # `systemctl daemon-reload`



Monitoraggio

- **Logging: lasciare una traccia dettagliata e persistente delle attività dei demoni**
 - principi generali
 - syslog tradizionale
 - rsyslog
 - syslog-ng
- **Diagnostica istantanea: comandi per sondare lo stato corrente di**
 - CPU
 - memoria
 - disco



Log di sistema

- I log (diari) tenuti dal sistema sono indispensabili per la diagnostica
 - anche per rilevare attività malevole o sospette
 - La loro stessa sicurezza va garantita!
 - Usare appropriatamente un integrity checker
 - Replicarli su macchine remote
- Logging su server remoto
 - Vantaggio aggiuntivo: centralizzazione
 - Implementazioni avanzate: shadow loggers
 - Problema: diventa un bersaglio appetibile
 - DoS



Linux logging

■ Soluzioni comuni

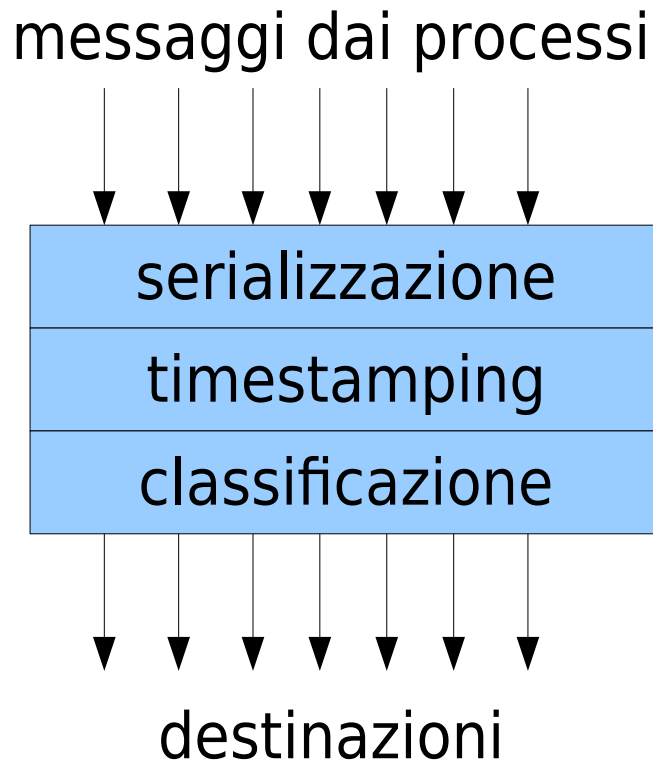
- Tipicamente producono file di testo
- Nessuna garanzia di uniformità di formato a parte la marcatura temporale
- BSD syslog (obsoleto)
 - klogd
- Rsyslog
- Syslog-ng

■ In prospettiva integrato in **systemd**

- *Journal*
- Attivo dal boot, non dipende dall'avvio di altri servizi
- Formato binario, visualizzabile con **journalctl**



- Principi di base mantenuti anche dalle evoluzioni



syslog: selettori e destinazioni

- Ogni messaggio è etichettato con una coppia *<facility>.<priority>*

- Facility = argomento

- auth, authpriv, cron, daemon, ftp, kern, lpr, mail, news, syslog, user, uucp, local0..local7

- Priority = importanza in ordine decrescente:

- emerg, alert, crit, err, warning, notice, info, debug

- Le destinazioni possibili sono

- **File**: identificato da path assoluto

- **STDIN** di un processo: identificato da una pipe verso il programma da lanciare

- **Utenti** collegati: username, o * per tutti

- **Server** syslog remoto: @indirizzo o @nome

- La comunicazione avviene di default su UDP, porta 514

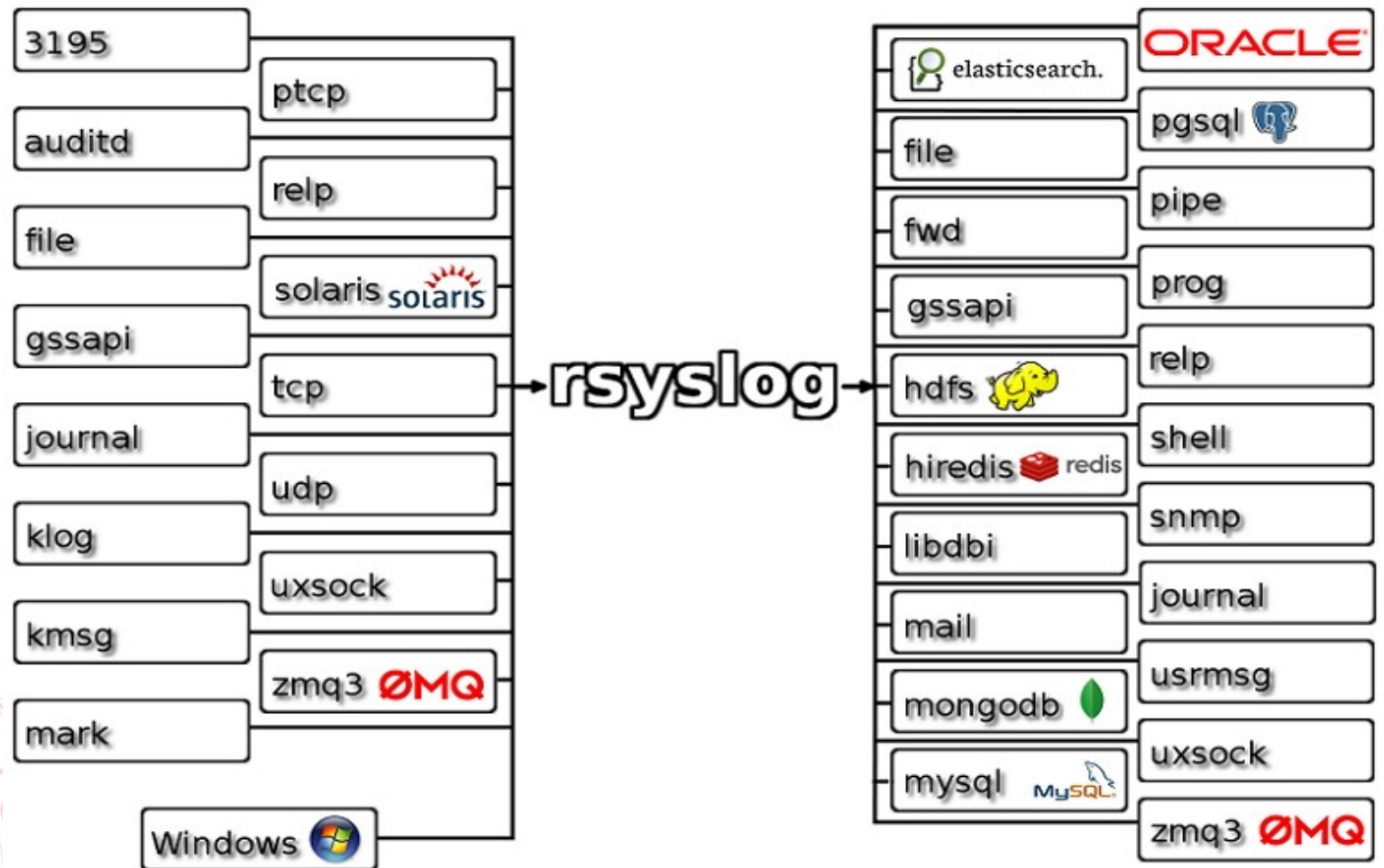
syslog: selettori

- **/etc/syslog.conf** contiene le direttive di configurazione generale e le regole minime di smistamento dei messaggi
- Ogni riga = una regola
 - **<etichetta di interesse>** **<destinazione>**
 - Vengono parsate tutte, quindi un messaggio può finire su più destinazioni
- **Trattamento delle priority**
 - Soglia: una regola che specifica una priority fa match con tutti i messaggi di tale priority e superiori a meno che non sia preceduta da “=”
 - Priority speciale *none*: serve per ignorare i messaggi con la facility specificata prima del punto

■ Es:

kern.*	/dev/console
*.info;mail.none;	/var/log/messages
*.emerg	*
kern.crit	" /usr/bin/alerter"
*.=warning	@loghost

rsyslog



rsyslog

■ Struttura modulare per caricare solo le funzioni necessarie

- es. attivazione della ricezione di messaggi via rete (v8.4 / v8-16):

```
$ModLoad imudp          /          module(load="imudp")  
$UDPServerRun 514        /          input(type="imudp" port="514")
```

- es. integrazione del kernel logging

```
$ModLoad imklog          /          module(load="imklog")
```

■ File di configurazione modulare

- Direttive globali in **/etc/rsyslog.conf**
- Direttive specifiche in file separati sotto **/etc/rsyslog.d/**
 - stessa sintassi

■ Scarto di messaggi (per evitare che vengano catturati da troppi selettori)

- Basta mettere ~ come destinazione



rsyslog – modalità di output evolute

■ Template per definire canali di output

- Possono sostituire le destinazioni in modo più flessibile

- Definizione:

```
$template  
apacheAccess, "/var/log/external/%fromhost%/apache/  
%msg:R,ERE,1,ZERO:imp:([a-zA-Z0-9\ -]+)\. --end%-access.log"
```

- Utilizzo:

```
local6.notice ?apacheAccess
```

Segnaposto che verrà
sostituito dal nome
dell'host che origina
il messaggio

Segnaposto che verrà
sostituito per mezzo di
una elaborazione del
messaggio fatta via regex

■ TCP logging

- Per evitare perdita di messaggi (finché non ci sono crash!)

<http://blog.gerhards.net/2008/04/on-unreliability-of-plain-tcp-syslog.html>

```
*.* @@indirizzo
```

■ Shell execution

- Passa il messaggio come parametro a un programma

```
*.* ^programma;template
```

rsyslog – selettori evoluti

- Rsyslog seleziona i messaggi in tre modi
 - i tradizionali facility.priority
 - Filtri basati su proprietà
 - Filtri basati su espressioni
- Filtri basati su proprietà
 - **:property, [!]compare-operation, "value"**
 - Es: **:msg, !contains, "error" /var/log/good.log**
- Filtri basati su espressioni
 - Ancora in evoluzione (a marzo 2021)
 - la sintassi potrebbe cambiare in futuro
 - **if expr then destinazione**
 - Diventeranno un sistema completo di scripting, che consentirà di eseguire programmi arbitrari per determinare la destinazione



rsyslog – moduli

- Troppi per citarli esaustivamente

<http://www.rsyslog.com/doc/v8-stable/>

- Tra i più interessanti:

- RELP logging - per garanzia totale di consegna

```
$ModLoad omrelp
```

```
*.* :omrelp:indirizzo
```

- Output su tabelle di database

```
$ModLoad ommysql
```

- Acquisizione diretta dei messaggi del kernel (sostituisce klogd)

```
$ModLoad imklog
```



syslog-ng

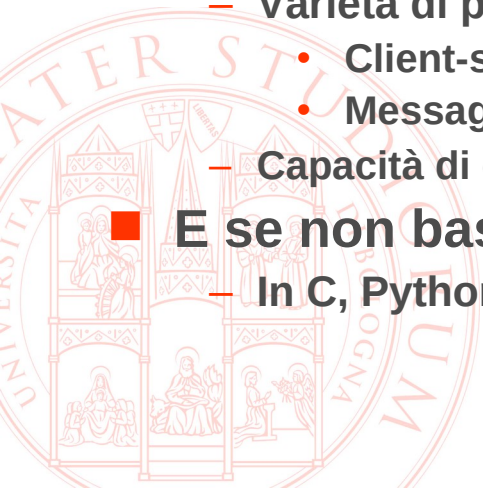
<https://syslog-ng.org/>

■ Flessibile

- Input compatibile con
 - Formati standard syslog (RFC3164, RFC5424)
 - JSON
 - Journald (systemd)
- Output verso molteplici destinazioni
 - Tutti i DB SQL più diffusi
 - DB NOSQL (es. MongoDB)
 - Cloud databases (es. Redis)
- Varietà di protocolli
 - Client-server
 - Message-based (AMQP, STOMP)
- Capacità di elaborazione del contenuto dei messaggi

■ E se non basta, estendibile con plugin

- In C, Python, Java, Lua, o Perl



syslog-ng

■ Configurazione:

- Definizione di una *source*, che può unificare più ingressi fisici

```
source s_two {  
    network(ip(10.1.2.3) port(1999));  
    network(ip(10.1.2.3) port(1999) transport("udp"));  
};
```

- Definizione di una *destination*, con relative opzioni

```
destination d_file {  
    file("/var/log/${YEAR}.${MONTH}.${DAY}/messages"  
    template("${HOUR}:${MIN}:${SEC} ${TZ} ${HOST} [${LEVEL}] ${MSG}\n")  
    template-escape(no));  
};
```

- Attivazione di un canale di log

```
log { source(s_two); destination(d_file); };
```

Monitoraggio dei parametri di sistema

■ Comandi essenziali per il monitoraggio

(Utenti)	File	Processi	Spazio
(w) (last)	fuser	ps top uptime	df du free
	lsof		vmstat iostat

■ La maggior parte sono interfacce verso il filesystem **/proc**

<https://www.kernel.org/doc/html/latest/filesystems/proc.html> →

<https://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html>

proc filesystem

- Uno pseudo-filesystem che mostra come file i parametri di funzionamento del sistema
- Non risiede su disco, è una “apparizione” generata dal sistema operativo attraverso il meccanismo dei virtual filesystem
 - l’esplorazione della gerarchia viene gestita mostrando un’organizzazione di file e cartelle corrispondenti alle strutture dati del kernel
 - le chiamate in lettura a un file sono gestite mostrando i dati, presenti nelle strutture di memoria del kernel, corrispondenti al file utilizzato
 - sono presenti anche file scrivibili: inserirvi dati equivale a riconfigurare istantaneamente i corrispondenti parametri di funzionamento del kernel
- Parti principali
 - **directory con nomi numerici** corrispondenti ai PID dei processi attivi
 - **directory** che rappresentano alcuni macro-sistemi **hardware**
 - acpi, bus, driver, irq, tty
 - **directory** che rappresentano parametri o funzioni del **sistema operativo**
 - fs, sys, sysvipc
 - **file** con informazioni (principalmente **statistiche** di uso) globali del sistema

ps

- **ps (1)** supporta un numero strabiliante di opzioni, perché è compatibile con ben tre sintassi
 - Unix, singole lettere precedute da singolo trattino
 - BSD, singole lettere, senza trattino
 - estensioni GNU, parole precedute da doppio trattino
- Le opzioni delle tre famiglie possono essere mescolate nello stesso comando, a meno di non creare contraddizioni o ambiguità (...)
- Per gli usi più comuni, ci sono esempi collocati all’inizio della man page
- Suggerimenti – andiamo a leggere la man page
 - la sezione **PROCESS SELECTION BY LIST** mostra come ottenere una lista di processi secondo le loro proprietà (es. comando lanciato, pid, utente, ecc.)
 - è molto meglio usare queste opzioni che non “greppare” l’intera lista di processi!
 - la sezione **OUTPUT FORMAT CONTROL** illustra come formattare la lista prodotta
 - in particolare le opzioni (equivalenti) **-o, o, --format** seguite da una stringa di specificatori documentata nella sezione **STANDARD FORMAT SPECIFIERS** permettono un controllo completo sui campi che si vogliono far comparire nella lista
 - la sezione **PROCESS STATE CODES** spiega il significato della colonna STAT e dà un’indicazione fondamentale dello stato del processo

uptime

- uptime prende il nome dal primo elemento di output
- riporta anche il *carico* del sistema
 - ad ogni invocazione dello scheduler viene registrato il numero **totale** di processi in stato R (runnable) o D (uninterruptable sleep)
 - i campioni vengono accumulati e mediati su tre scale temporali diverse per ottenere un'indicazione del trend nel tempo
 - lo stato di “salute” va valutato in confronto al numero di processori disponibili

21:27:56 up 7:10, 2 users, load average: 0.00, 0.00, 0.00

ora n. utenti connessi

carico medio negli ultimi...

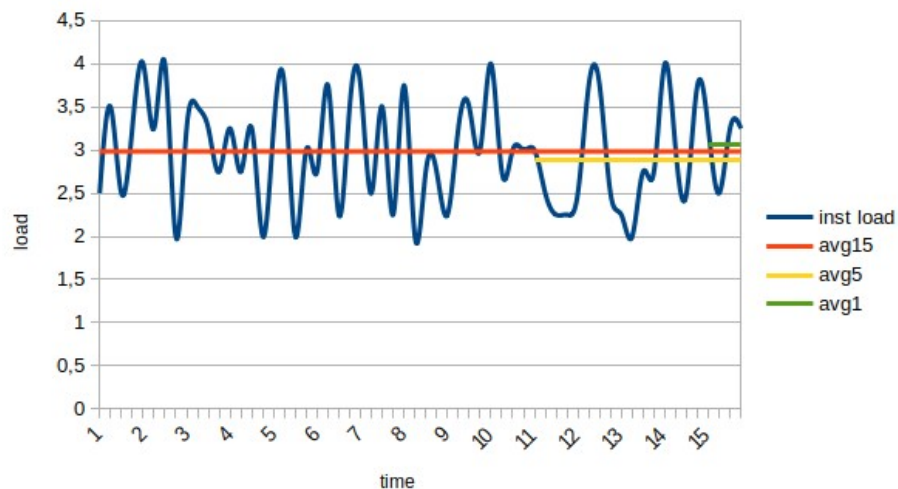
tempo trascorso dal boot

1' 5' 15'

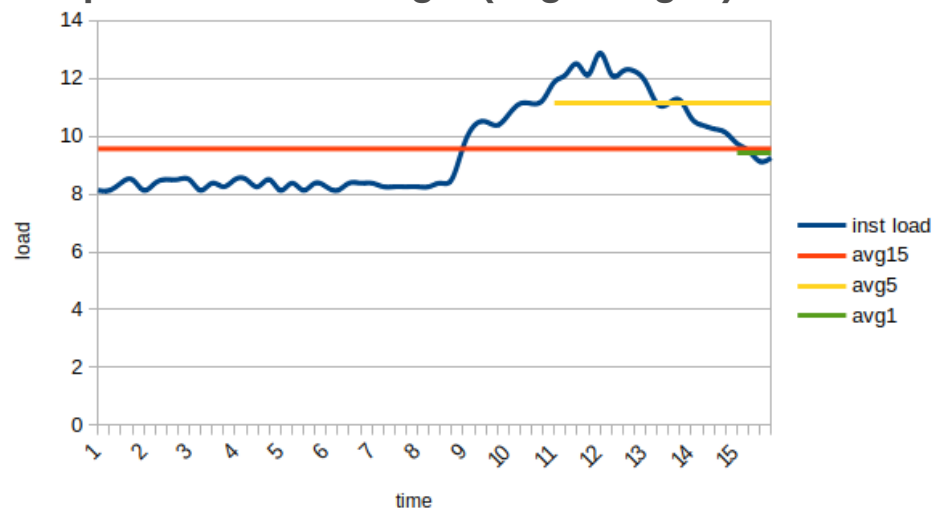


i tre carichi di uptime: esempi di interpretazione

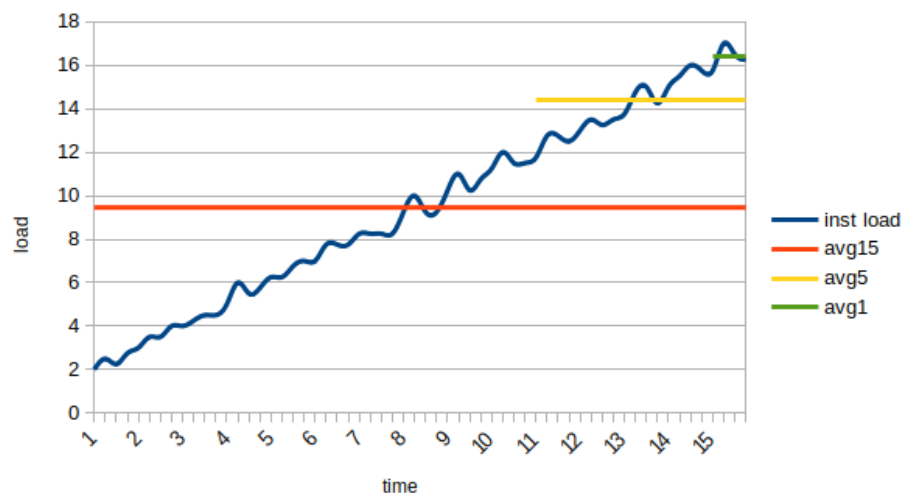
carico costante: $\text{avg1} \approx \text{avg5} \approx \text{avg15}$



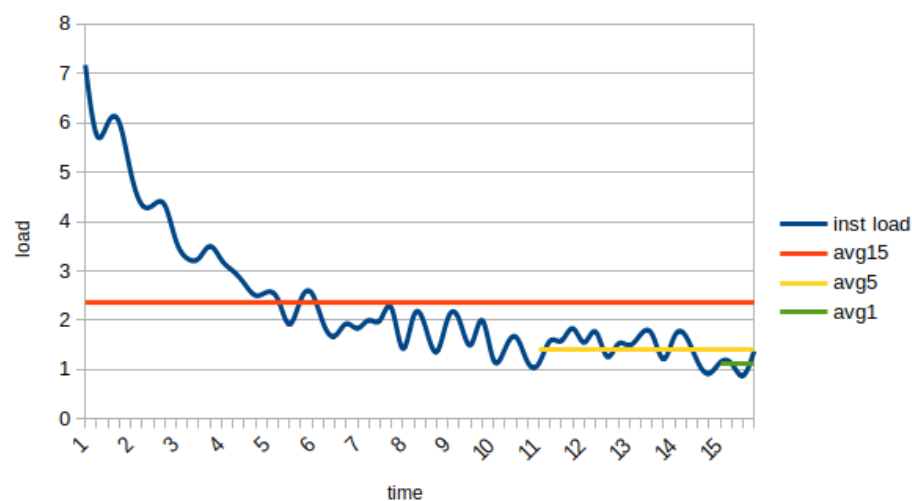
picco recente: $\text{avg5} > (\text{avg1} \approx \text{avg15})$



carico crescente: $\text{avg1} > \text{avg5} > \text{avg15}$



carico calante: $\text{avg1} < \text{avg5} < \text{avg15}$



free

las@client:~\$ **free**

	total	used	free	shared	buff/cache	available
Mem:	237040	121248	9596	1464	106196	98720
Swap:	1045500	29572	1015928			

- la maggior parte della memoria usata per cache può essere liberata per usi prioritari, da cui **available \approx free + buff/cache**
 - l'impatto sulle prestazioni della rinuncia alle cache non è nullo
- **used swap > 0** significa solo che in qualche momento è servita



ps – uptime – free → top

■ Comandi che scattano un'istantanea del sistema

- **ps**: stato dei processi
- **uptime**: carico del sistema
- **free**: occupazione memoria

■ Comandi di monitoraggio interattivi

- **top** riassume ps, uptime, free + **uso dettagliato cpu**
- aggiornato regolarmente
- permette di interagire coi processi
- utile per stima intuitiva dello stato di salute



top

9:31am up 50 min, 2 users, load average: 0.02, 0.02, 0.04

71 processes: 70 sleeping, 1 running, 0 zombie, 0 stopped

CPU states: 4.3% user, 5.2% system, 0.1% nice, 90.2% idle

Mem: 384480K av, 380688K used, 3792K free, 1312K shrd, 51312K buff
Swap: 128516K av, 0K used, 128516K free 139136K cached

PID	USER	PRI	NI	SIZE	RSS	SHARE	STAT	%CPU	%MEM	TIME	COMMAND
1179	root	13	0	3092	3092	2592	S	2.8	0.8	0:50	magicdev
9299	root	16	0	1044	1040	832	R	2.8	0.2	0:00	top
1	root	8	0	520	520	452	S	0.0	0.1	0:03	init
2	root	9	0	0	0	0	SW	0.0	0.0	0:00	keventd
3	root	9	0	0	0	0	SW	0.0	0.0	0:00	kapm-idled
4	root	19	19	0	0	0	SWN	0.0	0.0	0:00	ksoftirqd_CPU0
5	root	9	0	0	0	0	SW	0.0	0.0	0:00	kswapd
6	root	9	0	0	0	0	SW	0.0	0.0	0:00	kreclaimd
7	root	9	0	0	0	0	SW	0.0	0.0	0:00	bdflush
8	root	9	0	0	0	0	SW	0.0	0.0	0:00	kupdated
9	root	-1	-20	0	0	0	SW<	0.0	0.0	0:00	mdrecoveryd
71	root	9	0	0	0	0	SW	0.0	0.0	0:00	khubd
465	root	9	0	0	0	0	SW	0.0	0.0	0:00	eth0
546	root	9	0	592	592	496	S	0.0	0.1	0:00	syslogd
551	root	9	0	1124	1124	448	S	0.0	0.2	0:00	klogd
569	rpc	9	0	592	592	504	S	0.0	0.1	0:00	portmap
597	rpcuser	9	0	788	788	688	S	0.0	0.2	0:00	rpc.statd

top – esempi di interpretazione del carico

- Un carico elevato può essere dovuto a molte cause diverse
- Esaminare l'uso di memoria e CPU può dare un'indicazione
- Es.
 - CPU sostanzialmente scarica
 - molti processi D? → un dispositivo non risponde?
 - CPU usata principalmente in userspace
 - sistema CPU-bound
 - CPU usata principalmente in iowait
 - sistema I/O bound
 - possono essere periferiche lente ma anche sovraccaricate per altri motivi
 - swap molto usata? → disco bombardato di swapout-swapin
 - sistema memory-bound
- indagini più approfondite si possono svolgere con **vmstat** e **iostat**
 - idealmente disponendo di una **baseline**, cioè dei valori tipici misurati durante la condizione di uso ottimale del sistema

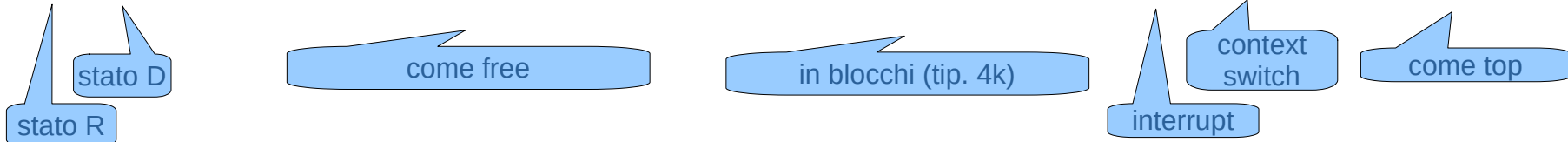


Evoluzione delle risorse

- **vmstat** – uso di memoria, paging, I/O, trap
 - utile invocarlo col periodo (in secondi) per monitorare

```
root@Client:~# vmstat 1
```

procs		memory				swap		io		system		cpu				
r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa	st
0	0	0	53404	39008	89588	0	0	12	1	12	31	0	0	100	0	0
0	0	0	53344	39008	89588	0	0	0	0	19	27	0	1	99	0	0
0	0	0	53344	39008	89588	0	0	0	0	14	19	0	0	100	0	0



- **iostat** - statistiche su uso CPU e I/O
 - soprattutto per valutare l'uso dei dispositivi di I/O

```
root@Client:~# iostat /dev/sda1
```

```
Linux 3.16.0-4-amd64 (Client) 03/21/18 _x86_64_ (1 CPU)
```

Device:	tps	kB_read/s	kB_wrtn/s	kB_read	kB_wrtn
sda1	0.65	11.62	0.81	123899	8668

Spazio disco

- **df** mostra l'utilizzo dello spazio disco:

```
$ df
Filesystem      1K-blocks    Used Available Use% Mounted on
udev             101812         0      101812   0% /dev
tmpfs            23704        3236       20468  14% /run
/dev/sda1       19277044 1654936  16619820  10% /
tmpfs           118520         0      118520   0% /dev/shm
tmpfs            5120          0         5120   0% /run/lock
tmpfs           118520         0      118520   0% /sys/fs/cgroup
tmpfs           23704          0       23704   0% /run/user/1001
```

- **du** permette di calcolare lo spazio occupato dai file (in una directory). Senza opzioni particolari du riporta l'occupazione totale delle dir passate come argomento ed anche di tutte le subdir in esse presenti. Es:

```
# du /tmp
1 /tmp/.font-unix
1 /tmp/.X11-unix
1 /tmp/.ICE-unix
5 /tmp/orbit-root
72 /tmp
```

- **du -s** riporta invece il *summary*, senza dettagli sulle subdir.

Uso dei file

■ Quali file sta usando un processo:

```
# ls -l /proc/2208/fd/
total 0
lrwx----- 1 root    root    64 Apr 26 10:11 0 -> /dev/pts/0
lrwx----- 1 root    root    64 Apr 26 10:11 1 -> /dev/pts/0
lrwx----- 1 root    root    64 Apr 26 10:11 2 -> /dev/pts/0
lr-x----- 1 root    root    64 Apr 26 10:11 3 -> /etc/man.config
```

■ Quali processi stanno usando un file:

```
# fuser /etc/man.config
/etc/man.config: 2208 2212 2213 2219
```

– o un intero filesystem:

```
# fuser -m /var
/var: 546 597c 714 714c 879c 898 916
916c 964 1013 1020 1021 1318 6493 9244 9244m 9249 9249m
9275
```

c current directory.
e executable being run.
f open file. f is omitted in default display mode.
r root directory.
m mmap'ed file or shared library.

Uso globale dei file

■ **lsuf** – list open files

- elenca tutti i file impegnati da tutti i processi

■ opera su tutti i namespace riconducibili al concetto astratto di file

- regular file
- directory
- block special file,
- character special file
- executing text reference
- library
- stream o network file (socket internet o UNIX domain, NFS file)

■ Osservazione: un file cancellato (unlink) dopo l'apertura sarà irreperibile sul filesystem, ma referenziato dal processo e quindi visibile a lsuf



lsof

COMMAND	PID	TID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
cat	3210		las	cwd	DIR	8,1	4096	920949	/home/las
cat	3210		las	rtd	DIR	8,1	4096	2	/
cat	3210		las	txt	REG	8,1	43744	655603	/usr/bin/cat
cat	3210		las	mem	REG	8,1	3031632	654527	/usr/lib/locale/locale-archive
cat	3210		las	0u	CHR	136,1	0t0	4	/dev/pts/1
cat	3210		las	1w	REG	8,1	0	922420	/home/las/ghostfile (deleted)
cat	3210		las	2u	CHR	136,1	0t0	4	/dev/pts/1
lsof	3220		las	cwd	DIR	8,1	4096	920949	/home/las
lsof	3220		las	rtd	DIR	8,1	4096	2	/
lsof	3220		las	txt	REG	8,1	175584	666005	/usr/bin/lsof
lsof	3220		las	mem	REG	8,1	3031632	654527	/usr/lib/locale/locale-archive
...									

